High-Performance Testing

Executives and managers, get your performance testing teams out of the pit and ahead of the pack

By Scott Barber

As an activity, performance testing is widely misunderstood, particularly by executives and managers. This misunderstanding can cause a variety of difficulties—including outright project failure. This article details the topics that I find myself teaching executives and managers time and time again. Learning, understanding, and applying this knowledge on your performance testing projects will put you on the fast track to success.

Executives—Start your Engines . . .

Insist on Experience

Experienced performance testers will speak your language and guide you through the process of meeting your goals, even if you can't yet verbalize those goals. Experienced performance testers not only know how to relate to executives in terms of business risks, short- and long-term costs, and implications of schedule adjustment but they also know how to explain their trade without all the jargon and techno-babble. Experienced performance testers are used to explaining the relevance of the latest "performance buzz" to your system. They have spent years learning how to extract performance goals from such words as "fast," "maximum throughput," and "xxxx concurrent users"—none of which has meaning in isolation.

Jason Dewey/Stone/Getty Images

Consider this example: An executive dictates that, "Each page will display in under x seconds, 100 percent of the time." While this is both quantifiable and testable, it has no meaning on its own. It is the job of the performance tester to define the conditions under which the goal applies, in other words, to determine the goal's context. To have meaning, this goal must address such things as client connection speed, number of people accessing the site, and types of activities those people are performing. These are all variables that affect page response time. A more complete goal would take this form:

Using the "peak order" workload model, under a 500-hourly user load, static pages will display in under x seconds, dynamic pages in under y seconds, and search/order pages in under z seconds, 95 percent of the time with no errors when accessed via corporate LAN.

Experienced performance testers also know how to collect and present data in ways that show whether the system is missing or meeting goals, in what areas, and by how much, without requiring the viewer of this data to have any special technical knowledge of the system under test.

Notice that I use the term "goal" instead of "requirement" when speaking about performance. I do this because I have never been involved in a performance testing project that delayed or canceled a release due to performance test results not meeting the stated criteria. I also choose the term "goal" because virtually no one expects the performance to be as good as he wants prior to Release One. What people hope for is "good enough for now." There is an assumption that performance will be improved during testing, that the production environment will resolve the performance issues detected in the test environment, and/or that adoption will be gradual enough to deal with performance problems as they arise in production. An experienced performance tester will be able to help you convert your feelings about performance into goals and project plans. Above all, performance testers want you, the executive, to understand performance testing so you can make sound, informed decisions. As an executive, you have several important decisions to make about an application during the development lifecycle. Most of the decisions center around three fundamental questions:

- Does it meet the need/specification for which it was developed?
- Will the application function adequately for the users of the system?
- Will the user of the system be frustrated by using it?

The experienced performance tester knows the importance of these questions—and their answers—and will work with you (literally by your side at times) to help you answer the questions in terms of performance.

TAKE THE LEAD . . .

First and foremost, you must make it known that you expect experienced performance testers on your projects, not "fools with tools" as some folks refer to them. Set the expectation early that the performance tester is expected to interact with you, and that his job is to provide you with the information you need to make sound business decisions about performance issues and risks. Always make a point to personally review performance goals to make sure they contain enough context to make them meaningful for executive-level decision making.

Review the performance test plan and deliverables and ask yourself the following questions:

- Will this assist with "go-live" decisions?
- Is it likely that the results from this plan could lead to a better experience for the end-user?
- Is this likely to be representative of the actual production environment?
- Is this likely to be useful to developers if tuning is necessary?
- Will it provide an answer to each specific requirement, goal, or service level agreement?
- Is taking action based on the results part of the plan?

Finally, invite the performance tester to educate you along the way. In helping you to expand your knowledge about performance testing, the tester will gain a wealth of knowledge about what is most important to you in terms of performance. This mutual understanding and open communication are the best things that can happen to the overall performance of a system.

Begin Performance Testing Before the Application Is Fully Functional

There is a common perception that performance testing can't effectively start until the application is stable and mostly functional—meaning that performance test data won't be available until significantly into a beta or qualification release. This leaves virtually no time to react if, or more realistically when, the results show that the application isn't performing up to expectations.

In actuality, an experienced performance tester can accomplish a large number of tasks and generate a significant amount of useful data even before the first release to the functional testing team. He can create and gain approval of User Community Models and test data and can gather these kinds of statistics:

- Network and/or Web server throughput limits
- Individual server resource utilization under various loads
- Search speeds, query optimization, table/row locking, and speed versus capacity measurements for databases
- Load balancing overhead/capacity/effectiveness
- Speed and resource cost of security measures

Some developers and system architects argue that the majority of these tasks belong to them, but developers rarely have the ability to generate the types of load needed to complete the tasks effectively. Adding the performance tester to these tasks early on will minimize the number of surprises and provide foundational data that will greatly

speed up the process of finding root causes and fixing performance issues detected late in the project lifecycle.

TAKE THE LEAD . . .

This one is pretty obvious. Plan to have a performance tester assigned to the project from kickoff through roll out. Encourage the development team to use the tester's skills and resources as a development tool, not just as a post-development validation tool. It is worth noting that, depending on the project, the performance tester is used for performance-related activities between 50 and 100 percent of the time. The upside is that, because of the skillset noted in the "Skills and Experience" sidebar, this individual can be fully utilized as a supplemental member of virtually every project team. There is one caveat: Make it clear that performance testing is this person's primary responsibility—not an additional duty. This distinction is critical because "crunch time" for performance testing typically coincides with "crunch time" for most of the other teams with which the performance tester may be working.

Don't Confuse Delivery with Done

"Delivery" is an informed decision based on risks and should not be confused with "done." Anyone who has been around testing for a while knows that the system will be deployed when management thinks holding up the release is riskier than releasing it, even if that means releasing it with unresolved or untested performance issues. However, releasing the software is no reason to stop performance testing.

Most applications have a rollout plan or an adoption rate that ensures that the peak supported load won't occur for a significant period of time after the go-live day. That is prime time to continue performance testing. There are fewer distractions, the existence of actual live usage data rather than predictions or estimations, the ability to observe performance on actual production hardware, and often the availability of more resources. If there isn't a maintenance release scheduled soon enough to get the post-release fixes into production before usage reaches the performance limit, surely it's more cost effective to schedule one than to contend with a performance issue when it presents itself in production.

TAKE THE LEAD . . .

Plan to continue performance testing after the initial release. Plan to push maintenance releases with performance enhancements prior to the first expected load peak. Incorporating these plans into the project plan from the beginning allows you to release software when performance is deemed acceptable for early adopters rather than holding up releases until the performance is tuned for a larger future load.

Test Managers—Start your Engines . . .

Look for Skills and Experience

Quality performance testers are senior members of the project team in terms of depth and breadth of skills and experience. You probably have read similar statements before, but I assure you that this isn't "the same ol' message in a new suit." When asked, "What skills should I look for in a performance testing candidate?" I reply, "What you want is a mid-level everything." The "Skills and Experience" sidebar lists specific skills to look for in a performance tester.

TAKE THE LEAD . . .

This really comes down to employee/consultant selection and training on which you likely have significant influence. In my experience, a top-notch performance testing candidate should be able to field many to most of the questions that an interviewer would ask of a mid-level developer/DBA/systems administrator. Obviously, this is not the expectation for most functional testing candidates. For instance, when interviewing a functional test candidate, you may ask questions such as, "How comfortable are you working directly with code?" When interviewing a performance test candidate,

SKILLS AND EXPERIENCE

Performance testing requires a unique skillset that is above and beyond what is generally expected for members of a functional testing team and in many cases more broad than what is generally expected of development team members. In addition to expertise with their tool(s) of choice, I specifically look for the following experience and skills (roughly listed in decreasing order of importance):

- User community modeling and simulation
- Communication protocol of system under test (i.e., HTTP/S for Web applications)
- Networking/Network architecture (i.e., MCSE for Microsoft networks)
- Statistics (for meaningful presentation and interpretation of data)
- Graphical presentation of complex information
- Application exploration, such as asking, "What if I do X while someone else does Y and twenty people do Z, all right before the nightly back-up kicks off?"
- Reliability, stability, and security testing
- System monitoring/administration (i.e., memory usage, disk I/O, etc.)
- Functionality testing
- Skill/experience as a trainer and/or facilitator
- Business analysis
- Human factors/usability studies
- Programming languages (specifically the languages of the load generation tool and the application under test)
- Database design/administration

Note that I don't mean for this list to be exhaustive. These are the skills that I feel are "required" for one to be considered a senior performance tester over a range of software projects.

it is completely appropriate to ask questions such as, "What was the most complex custom function you ever had to code to enable your load generation script to accurately represent the expected usage scenario? What made the function complex? Do you still have the code and/or could you re-create it easily?"

Select Testers Who Know Their Tools

Quality performance testers need quality tools—and need to know how to use them. This is the closest thing to a universal rule I advocate. The only way to conclusively determine application performance under load is to put load on the system. While it is possible, and under certain circumstances valuable, to use what I refer to as the "hire a bunch of interns" method to generate load on a system, this method makes it almost impossible to replicate scenarios that uncover performance issues.

I am not aware of a way to generate reproducible load without some kind of load generation/simulation tool. Whether that tool is purchased, open source, or custom built is basically irrelevant. What matters is that load scenarios are reproducible, modifiable, and match predetermined user community models.

This may sound obvious to some, but my experience is that it's not as obvious as it should be. No matter what a tool can do in theory, it actually only does what it's "told" to do. A good performance tester with expertise using a "non-first-tier" tool of his choice will almost always be able to create a more effective performance test and generate more accurate and useful results faster than the best performance tester with a new or unfamiliar tool.

Selecting a tool because of a vendor relationship, market review, or advertisement will not immediately lead to high-quality performance testing—no matter how good the tool is. I doubt many would debate that the single most important measure of the quality of a tool is your performance tester's ability to use it to generate useful, accurate, and timely results.

Plan to use a load generation tool as part of your performance testing. If a performance test strategy is proposed that does not include the use of a tool, ask how tests will be validated, repeated, and modified to isolate observed performance issues. Only if those questions are answered to your satisfaction should you consider approving a strategy that does not include the use of a load simulation/generation tool.

IN ACTUALITY, AN EXPERIENCED PERFORMANCE TESTER CAN ACCOMPLISH A LARGE NUMBER OF TASKS AND GENERATE A SIGNIFICANT AMOUNT OF USEFUL DATA EVEN BEFORE THE FIRST RELEASE TO THE FUNCTIONAL TESTING TEAM.

TOOL-DRIVEN TEST DESIGN

Consider this. When people design their dream homes, do they start by taking inventory of their toolboxes and then design in only those features that they can build with their tools? Of course not. They design their homes, then determine what tools or contractors they need to obtain or hire. Only if they are unable to obtain the proper tools or cannot afford an appropriate contractor do they modify their designs with an acceptable alternative. The same concept should be applied to performance test design.

The most common mistake made by managers and testers alike is what I refer to as "tool-driven test design." Having agreed (I hope) that effective performance testing requires a load generation/simulation tool, we need to beware of building tests around that which is either easy to do using the tool or that which the vendor claims the tool does well. All of the tools on the market—including open source tools—were developed either to test a wide variety of applications or to test one specific application. But either way they were not developed to test your application—which demonstrates that what the tool facilitates may or may not apply to your specific application or your testing goals.

TAKE THE LEAD . . .

As a manager or executive, you have a key role in obtaining both a tool and a performance tester. I hope that it is clear that you should either hire the performance tester first, then provide that person with the tool he recommends for the job; or select a tool, then hire a performance tester with significant experience and success using that tool.

To avoid the "tool-driven test design" trap (see the sidebar "Tool-Driven Test Design"), don't implicitly trust the performance tester. Even the best performance testers can develop blind spots when it comes to their "pet tool." I have been guilty of that myself on a few occasions, one of which led to sixty hours of work between 6:00 p.m. Friday and 8:00 a.m. Monday to correct my oversight. Just to be safe, during the test design process instruct those involved to "Forget about the tool for now and just design the test. We'll worry about how we are going to implement the design later." After that, listen and observe. If you hear things such as "but the tool can't . . . " or "that will take too long with the tool . . .", remind the team of the initial instructions and make a note to discuss acceptable alternatives with the performance tester at a later time.

Get the Facts to Back Up the Predictions

Extrapolating production loads from data collected on test systems is at best "black magic." There are a few, very specialized individuals who can make these predictions with any degree of confidence or accuracy. These individuals, who are rarely performance testers, tend to project

(Continued on page 50)

(Continued from page 36)

the performance based on rather complex mathematical models that they began building prior to the first line of code ever being written. Anyone else who claims that he can predict the performance of an application on production hardware based on the results of testing on test hardware that is not virtually identical is likely waving a magic wand of estimation and assumption. It isn't accurate to say, for example, that the production environment has twice as many servers, so it will support twice as many total users. Without actual data to go on, it is equally as likely that doubling the number of servers will degrade performance rather than improve it.

TAKE THE LEAD . . .

Always demand to know the assumptions and review the calculations and data that contribute to any predictions. More importantly, ensure that part of your project plan involves at least one load test on production hardware. With proper planning a performance test that verifies predictions and/or identifies implementation errors typically takes only two to four hours. I often have seen those hours be the most valuable risk mitigation measure of the entire project.

The Finish Line

From the start, executives and managers set the pace for the performance testing component of their projects. This responsibility carries with it a need to be educated about performance testing, what it is, what it isn't, and what constitutes reasonable expectations. This article has provided an educational foundation while hopefully making sense out of some of the most common misconceptions related to performance testing in the industry today. **{end}**

Scott Barber is the CTO for PerfTestPlus, Inc. His specialty is context-driven performance testing and analysis for distributed multi-user systems. Contact him at sbarber@perftestplus.com.

(Continued from page 52)

what we do with our emotions. With self-regulation, we can choose how to express our emotions instead of letting our emotions control us. Self-regulation also means we can manage our moods, not just stew in them. Pat may be in a bad mood, but she doesn't need to take it out on those around her.

Social Skills

Social skills help us empathize with other people, see others' points of view, and share their concerns. Social skills help us build bonds, collaborate with others, and create a friendly atmosphere—all good things for the group with which you work. Pat isn't building any bonds with Marc or Jenny and is depriving herself of their insight, help, and expertise.

My friend's boss may think he's making a smart choice by pushing his staff solely to develop their technical skills. But he's likely paying a price in higher stress, lower productivity, and dampened work satisfaction. So if you really want to improve results, don't just pay attention to technical skills. As workers in a technical field, we're all pretty smart—and truly smart, successful people pay attention to the skills that help them work well with others. **{end}**

Esther Derby is one of the rare breed of consultants who blends the technical issues and the managerial issues with the people-side issues. She is well known for her work in helping teams grow to new levels of productivity. Esther coaches technical people who are making the transition to management and is a Certified Scrum Master. Her articles have appeared in Better Software *(formerly* STQE*),* Software Development, Cutter IT Journal, and CrossTalk. *She writes regular columns for StickyMinds.com and Computerworld.com, and publishes the quarterly newsletter, "insights".*

Have the Last Word!

If you have a point to make or a side to take on issues and trends that affect the industry, we want to hear from you.

Please send your submission to **editors@bettersoftware.com.**